Indexing Starts at 1
Default Units: SI

Roots & Zeros Sandbox:
Solving Non-Linear Single-Variable Equations

Example #1: Danger to yourself and others (Univ Physics 211 example)

You toss a rock into the air with an initial height above the ground of z0 with a given velocity (vo). Gravity pulls it down with an accelleration of g=-9.8 m s-2 (or 32 ft/s2)

The classic Physics formula is $z(t,g,z_o,v_o) := z_o + v_o \cdot t - \frac{1}{2} g \cdot t^2$

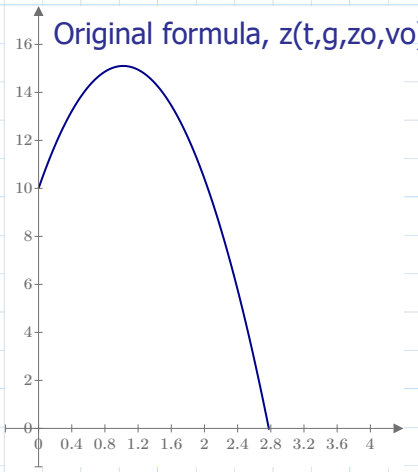
$g := 9.8$ m s-2 $\quad z_o := 10$ m $\quad v_o := 10$ m s-1 $\quad z_{target} := 2$ m

Warning! When playing with symbolic arrows for solving these root equations, take the units off your numbers! The above units are text boxes

When will it land on your foot or on your head? (where z = 0 for your foot, and z=2 m in some of our fantasies...)

This is a classic quadratic equation or a "polynomial" or "algebraic" equation. While very generic, if have to solve for eigenvalues in structures, Diff Eqs and other applications of eigenvalues, you'll find yourself with the same overall structure.

Original formula, z(t,g,zo,vo)



t_{plot} s

$z(t_{plot},g,z_o,v_o)$ m

In most cases you will want to plot out your original function. This will give you the lay of the land so that you can attack the problem.

Here we see that we have a classic parabolic shape (we could guess that from the equation) and luckilly it does pass through our desired value of 2 m.

But when (what value of t) specifically?

To better lock down the acutall target solutiuon in on our root value(s) of t, we will want to convert this into a "root-style" equation...

Indexing Starts at 1
Default Units: SI

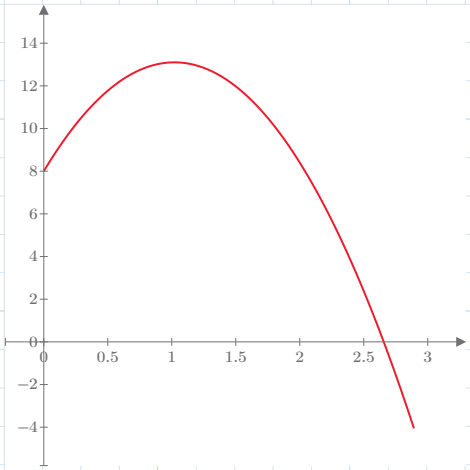
To get the root equation, no matter how trivial or scary the original equation, the approach is always the same...

Take the original equation, and slam ALL terms to one side of the equal sign. It doesn't matter whcih side. The result will be a "new" function so we should use a new label designator for it. If there are no "f"'s in the original equation, f(x) or here, f(t), is the traditional symbol to lead off the new function...

Root Equation Formula f(t)

$$f(t) := z_o + v_o \cdot t - \frac{1}{2} g \cdot t^2 - z_{target}$$

And our solution for where our object crosses our target height will be where f(t) = 0.



t s

$f(t)$ m

Let's make a new graph of our root function. Now we are able to see that it crosses the t-axis (where f(t) = 0) between 2.5 and 3.0 using the available tick marks that I can see.

These two values will represent our first guess(es).

Don't let any OCD get best of you when selecting brackets. Let the methods for getting the roots do the heavy lifting.

In Matchad, one of the easier functions to use with universal appeal is the **root**() function. It takes into account a bracket range which allows you to zero in on a solution without your solution "wandering" to values that you have already dismissed as non-viable. In this case, such a misdirection would send it to a negative time value and we don't want that!

$$t_{rootFunc} := \mathbf{root}\left(f(t), t, 2.5, 3\right) = 2.656 \quad \text{s}$$

We can test this by inserting that value of t into our original function or root function...

$$f\left(t_{rootFunc}\right) = 0 \,\text{m}$$

You should be satisfied if your answer is effectively zero within measurement or numerical precision.

Remember with Matchad you have about 15 or so floating point places so a value of 10^(-14ish to -15ish) is basically a defacto zero error.

Indexing Starts at 1
Default Units: SI

Let's now try the Solver Arrow. Be aware this doesn't always work. You'll see that happen in the Green-Ampt example that follows,

$$f(t) \xrightarrow{solve,\,t} \begin{bmatrix} -0.6147939284906228123 \\ 2.6556102550212350572 \end{bmatrix} \text{ m}$$

Here we ave two values, the negative time (already dismissed) and a value close to our result from **root** (and graph!). So we have an answer that is "Benching" with (matching) a previous approach. That's also good.

Note that if you play with the options under the Symbolic Area on the ribbon, you can refine your answer, and even bracket them...

$$t_{solveArrow} := f(t) \xrightarrow{solve,\,t,\,assume,\,t>0} 2.6556102550212350572 \text{ s}$$

$$t_{solveArrow} := f(t) \xrightarrow{solve,\,t,\,assume,\,t = RealRange(2.5,\,3)} 2.6556102550212350572 \text{ s}$$

Always test your results...

$$f\left(t_{solveArrow}\right) = 0 \quad \text{m}$$

Another satisfactory solution!

Since this is a polynomial expression we can use one more intrinsic function in Mathcad. This is the **polyroots**() function... (You can't do this with a transcendental function)

First we need to get the coefficients before each member of the polynomial expression. The "coeffs" arrow function will do this and sort them from the lowest-to-highest coefficient order, as shown here.

$$C := f(t) \xrightarrow{coeffs} \begin{bmatrix} 8 \\ 10 \\ -4.9 \end{bmatrix} \qquad f(t) \rightarrow 10 \cdot t + -4.9 \cdot t^2 + 8$$

$$t_{polyroot} := \text{polyroots}(C) = \begin{bmatrix} -0.615 \\ 2.656 \end{bmatrix} \text{s}$$

Grabbing the more likely candidate... (The positive one!)

$$f\left(t_{polyroot_1}\right) = 8.882 \cdot 10^{-16}$$

m

Indexing Starts at 1
Default Units: SI

Now on to programming. As we noted before, you can slide the "f" into the argument list so you can reuse the function after creating it. Here are three of our functions for root detection. First let's declare our programs (don't forget to also have an error function!)

$$error(a,b) := |a-b|$$

$$bisect(f, x_{lo}, x_{hi}, \varepsilon_{target}) := \left| \begin{array}{l} i \leftarrow 0 \\ x_{out} \leftarrow \frac{(x_{hi}+x_{lo})}{2} \\ x_{old} \leftarrow x_{hi} \\ \text{while } (error(x_{old}, x_{out}) > \varepsilon_{target}) \\ \quad \left| \begin{array}{l} x_{old} \leftarrow x_{out} \\ \text{if } (f(x_{out})\, f(x_{lo}) < 0) \\ \quad \left| x_{hi} \leftarrow x_{out} \right. \\ \text{else} \\ \quad \left| x_{lo} \leftarrow x_{out} \right. \\ x_{out} \leftarrow \frac{(x_{hi}+x_{lo})}{2} \\ i \leftarrow i+1 \end{array} \right. \\ \text{return } \begin{bmatrix} x_{out} \\ error(x_{out}, x_{old}) \\ i \end{bmatrix} \end{array} \right.$$

$$newton(f, x_{fg}, \varepsilon_{target}) := \left| \begin{array}{l} i \leftarrow 0 \\ x_{old} \leftarrow x_{fg} \\ x_{out} \leftarrow x_{old} - \frac{f(x_{old})}{f'(x_{old})} \\ \text{while } (error(x_{old}, x_{out}) > \varepsilon_{target}) \\ \quad \left| \begin{array}{l} x_{old} \leftarrow x_{out} \\ x_{out} \leftarrow x_{old} - \frac{f(x_{old})}{f'(x_{old})} \\ i \leftarrow i+1 \end{array} \right. \\ \text{return } \left(\begin{bmatrix} x_{out} \\ error(x_{out}, x_{old}) \\ i \end{bmatrix} \right) \end{array} \right.$$

$$secant(f, x_{fg1}, x_{fg2}, \varepsilon_{target}) := \left| \begin{array}{l} i \leftarrow 0 \\ x_{old} \leftarrow x_{fg1} \\ x_{m2} \leftarrow x_{fg2} \\ x_{m1} \leftarrow x_{fg1} \\ x_{out} \leftarrow x_{m1} - f(x_{m1}) \frac{(x_{m1}-x_{m2})}{f(x_{m1})-f(x_{m2})} \\ \text{while } (error(x_{old}, x_{out}) > \varepsilon_{target}) \\ \quad \left| \begin{array}{l} x_{old} \leftarrow x_{out} \\ x_{m2} \leftarrow x_{m1} \\ x_{m1} \leftarrow x_{out} \\ x_{out} \leftarrow x_{m1} - f(x_{m1}) \frac{(x_{m1}-x_{m2})}{f(x_{m1})-f(x_{m2})} \\ i \leftarrow i+1 \end{array} \right. \\ \text{return } \left(\begin{bmatrix} x_{out} \\ error(x_{out}, x_{old}) \\ i \end{bmatrix} \right) \end{array} \right.$$

You can borrow the "Bisection" program and insert the False Position equation build the False Position Method Program.

Indexing Starts at 1
Default Units: SI

Now let's test them using the same brackets and first guesses from above. For our tolerance, we can presume that we have a really good yardstick with markings to the nearest millimeter. Why length? That's becasue our units for the f(t) equation is for length (take a closer look at the f(t) equation from above)!.

$$\varepsilon_{tolerance} := \frac{0.001}{2} = 5 \cdot 10^{-4}\ \text{m}$$

Bisection Method

$$t_b := bisect\left(f, 2.5, 3, \varepsilon_{tolerance}\right) = \begin{bmatrix} 2.656 \\ 4.883 \cdot 10^{-4} \\ 9 \end{bmatrix} \begin{matrix} \text{s} \\ \text{m} \end{matrix}$$

And here we're using f(t) to get the error...

$$f\left(t_{b_1}\right) = -0.002$$

False Position (my function for this is hiding in the margin)

$$t_{fp} := falsepos\left(f, 2.5, 3, \varepsilon_{tolerance}\right) = \begin{bmatrix} 2.656 \\ 1.278 \cdot 10^{-4} \\ 3 \end{bmatrix} \begin{matrix} \text{s} \\ \text{m} \end{matrix}$$

$$f\left(t_{fp_1}\right) = 2.157 \cdot 10^{-4}$$

Secant Method

$$t_s := secant\left(f, 2.5, 3, \varepsilon_{tolerance}\right) = \begin{bmatrix} 2.656 \\ 3.7 \cdot 10^{-6} \\ 3 \end{bmatrix} \begin{matrix} \text{s} \\ \text{m} \end{matrix}$$

$$f\left(t_{s_1}\right) = 1.41 \cdot 10^{-8}$$

$$secant\left(f, 2.5, 2.6, \varepsilon_{tolerance}\right) = \begin{bmatrix} 2.656 \\ 1.411 \cdot 10^{-4} \\ 2 \end{bmatrix} \begin{matrix} \text{s} \\ \text{m} \end{matrix}$$

$$secant\left(f, 2.95, 3, \varepsilon_{tolerance}\right) = \begin{bmatrix} 2.656 \\ 1.671 \cdot 10^{-5} \\ 3 \end{bmatrix} \begin{matrix} \text{s} \\ \text{m} \end{matrix}$$

Newton-Raphson Method

$$t_n := newton\left(f, 2.5, \varepsilon_{tolerance}\right) = \begin{bmatrix} 2.656 \\ 2.037 \cdot 10^{-5} \\ 2 \end{bmatrix} \begin{matrix} \text{s} \\ \text{m} \end{matrix}$$

$$f\left(t_{n_1}\right) = -2.034 \cdot 10^{-9}$$

$$t_n := newton\left(f, 3, \varepsilon_{tolerance}\right) = \begin{bmatrix} 2.656 \\ 2.694 \cdot 10^{-4} \\ 2 \end{bmatrix} \begin{matrix} \text{s} \\ \text{m} \end{matrix}$$

$$f\left(t_{n_1}\right) = -3.557 \cdot 10^{-7}$$

You also can have some fun with the Newton method and explore how fast those the equation converges on the FIRST iteration.

$$t_{newton1} := 2.5 - \frac{f(2.5)}{f'(2.5)} = 2.664 \quad \text{s}$$

$$t_{newton1} := 3 - \frac{f(3)}{f'(3)} = 2.686 \quad \text{s}$$

This was pretty fast for both...

Why? Hint: Take a close look at the slope near our brackets in the root equation plot above.... The more linear the slope, the faster it will converge!

Indexing Starts at 1
Default Units: SI

Now let's look at a new Mathcad structure, the Solve Block

This structure is a partially isolated section of Matchad "code" that has a number of applications from solviong roots to solving complex differential equations.

The solve block is separated into three regions.

Guess Values

$z_o := 10\ m$
$z_{target} := 2\ m$
$v_o := 10\ m \cdot s^{-1}$
$g := 9.8\ m \cdot s^{-2}$

These are my constants (notice also that you can use units here - since these values will not be seen outside of the solve block to mess up other features like the solve arrows!)

$t := 0.5\ hr$

And here is the fiirst guess. Be aware that you may have to experiment with this!

I prefer to explicitly restate my constraining equations in my solve blocks so that everything is there for a reviewer to see. You can *use* the plain old f(z) = 0 but that will require the reviewer to scroll up and that can be annoying from their perspective.

Also if you want to use units, but weren't doing it before the block, you WILL want to restate the equation. Same thing if you are using new constants.

Constraints

$z_o + v_o \cdot t - \frac{1}{2} g \cdot t^2 - z_{target} = 0$

$t > 0$ And here I'm setting an additional to further limit my answer to a positive value

The find values will provide the answer for a given function in the constraint section.

Solver

$t_{block} := \mathbf{find}(t) = 2.656\ s$

And if you need to test the result, you can put the equation below the "Solver Area" but keep it in the solve block so you can make use of the constants used in the "Guess Area"

$z_o + v_o \cdot t_{block} - \frac{1}{2} g \cdot t_{block}^2 - z_{target} = 0\ m$

The first block is where you will want to keep constants that you don't want to mix with other parts of your sheet. What goes in this part of the block, STAYS in this part of the block, and is not seen outside of it by other equations or processes.

Also important here for many Solve Block applications is the "first guess" of the value(s) that you are trying to solve. Your first guess may influence your solution so you may need to experiment with this one if you get an answer that isn't what's in your intuitive ballpark.

The center section is for your constraints. These are the conditions for which your desired soliution should be valid. So for your root probem, your equation should use the conditional equal sign to zero. You can also include any other constraints to ensure that your result fits specific requirements, like time being positive as we are doing here.

The last section is the "Solver" where you get your desired answer. For roots and other applications like solving simultaneous equations, you can use the **find**() function.

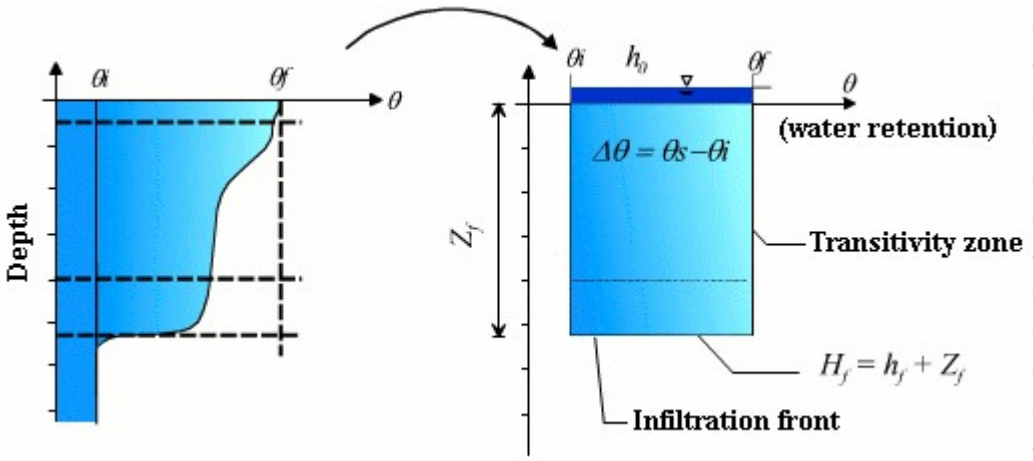
This final section will also let you export the value out of the solve block using the :=

$t_{block} = 2.656\ s$

For an example of how I comment these (and hint, hint, how I would like to see *you* comment on these structures), scroll down for the Green-Ampt example.

Indexing Starts at 1
Default Units: SI

Example #2: Hydrology - Infiltration of water into the soil



Consider a "front" of rainwater penetrating downward into the soil. (The term is acutally called the "Infiltration Front" or "Wetting Front".)

The depth and rate of infiltration can be expressed by the Green-Ampt equation where the infiltration depth as a function of the soil properties is....

$$depth := K_s \cdot t + \psi_s \cdot \Delta\Theta \cdot ln\left(1 + \frac{depth}{\psi_s \cdot \Delta\Theta}\right)$$

$$infiltration\_rate(depth) := K_s \cdot \left(\frac{\psi_s \cdot \Delta\Theta}{depth} + 1\right)$$

The Classic Green-Ampt Equations

Where for a Silt Loam Soil using soil property parameters estimated by Clapp and Hornberger (1978)

$K_s := 3.467 \cdot 10^{-5}$ m s-1 — Silt Loam Saturated Hydraulic Conductivity

$\psi_s := 0.218$ m — Silt Loam Saturated Matric ("Suction") Potential

$\Theta_s := 0.435$ — Silt Loam Saturation Water Content

Let's assume that the initial soil water content beneath the front (pre-rain) is

$\Theta := 0.4$ — Ambient Unsaturated Soil Water Content

$\Delta\Theta := (\Theta_s - \Theta) = 0.035$ — Wetting Front Deficit

After 10 minutes, let's estimate the the depth to which the water infulrated and estimate the infiltration rate

$t := 10 \cdot 60 = 600$ s

Indexing Starts at 1
Default Units: SI

To calculate the results of these two equations (normally we're really shooting for the infiltration rate), we want to first get the depth at time (t).

The depth equation on first glance is one that we can't solve with traditional Algebra-Fu. Both equations qualify as non-polynomial, non-algebraic and therefore is a "transcendental" equation.

(The second equation will just be an old-school "plug-n-chug" once we get the depth of the wetting front)

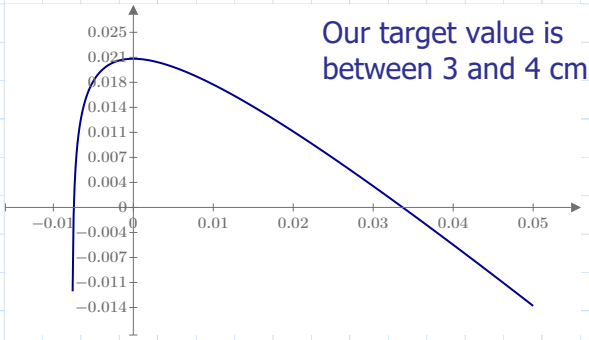
It is also an equation that typifies why we use the root method to solve these equations.

To solve this system, we first do what we do to all candidate root equatios. Send evetything to one side of the equal sign. (Once agian, we can use "f" as our symbol)

$$depth := K_s \cdot t + \psi_s \cdot \Delta\Theta \cdot ln\left(1 + \frac{depth}{\psi_s \cdot \Delta\Theta}\right)$$

$$f(depth) := K_s \cdot t + \psi_s \cdot \Delta\Theta \cdot \ln\left(1 + \frac{depth}{\psi_s \cdot \Delta\Theta}\right) - depth$$

This time to we can't make a reasonable plot of the first depth equation, but we CAN make the a plot of our root equation.

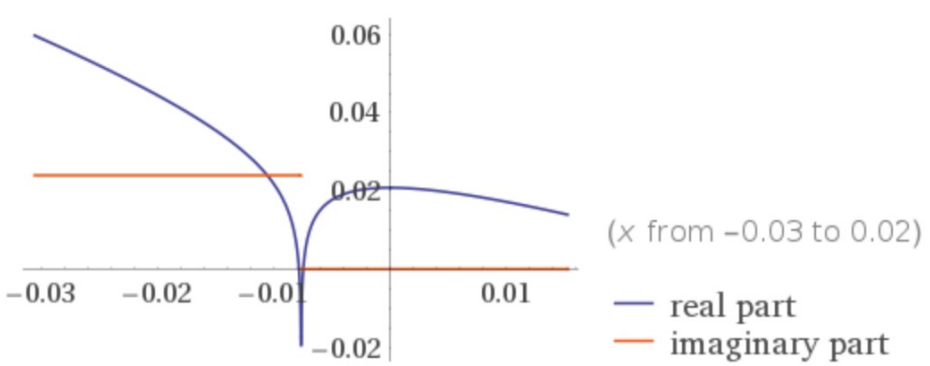


Our target value is between 3 and 4 cm

$f(depth)$ m

$depth$ m

Close inspection shows that at the far left end of our plot, we're running into that problem with the natural log part of the equation. That left side is hitting an asymptope. And the other side of that asymptope is an complex numnber with an imaginary component!



Indexing Starts at 1
Default Units: SI

So let's solve this using brackets of 3 cm and 4 cm
Here is our **root**() function solution...

$$d_{rootFunc} := \mathbf{root}\left(f(d), d, 0.03, 0.04\right) = 0.034 \text{ m}$$

$$f\left(d_{rootFunc}\right) = 1.117 \cdot 10^{-6}$$

This error is basically down to a micron!

Notice that the error here is small but bigger than our earlier case but still small enough for our purposes.

As I warned you, the solve arrow does not provide a working solution here. It gets the negative root of the system but can't seem to recognize our 3.4 cm solution.

$$f(d) \xrightarrow{solve, d} -0.007441672542940424657 \text{ m}$$

$$f(d) \xrightarrow{solve, d, assume, d = real} -0.007441672542940424657 \text{ m}$$

$$f(d) \xrightarrow{solve, d, assume, d > 0} ?$$

$$f(d) \xrightarrow{solve, d, assume, d = RealRange(0.03, 0.04)} ?$$

Indeed, I got desperate and even tried creating program variant of our root function to force the matter

$$g(d) := \left\| \begin{array}{l} \text{if } d > 0 \\ \quad \left\| g \leftarrow K_s \cdot t + \psi_s \cdot \Delta\Theta \cdot \ln\left(1 + \frac{d}{\psi_s \cdot \Delta\Theta}\right) - d \right. \\ \text{else} \\ \quad \left\| g \leftarrow 10000 \right. \\ \text{return } g \end{array} \right.$$

$$g(d) \xrightarrow{solve, d, assume, d = RealRange(0.03, 0.04)} ?$$

The solve arrow doesn't work with conditional expressions :-(

Honestly, we've stumped the prof here...

Indexing Starts at 1
Default Units: SI

Now for the code... Once again, our root dimensions are "Length." And let's have our units now calibrated for centimeters. (We're digging in dirt right now)

$$\varepsilon_{tolerance} := \frac{0.01}{2} = 0.005 \quad \text{m}$$

When we apply these functions to our brackets and first guesses.
Since we're working in dirt, I'm going to use wide brackets... (from surface to half a meter)

Bisection Method

And here we're using f(d) to get the error...

$$d_b := bisect\left(f, 0.0, 0.5, \varepsilon_{tolerance}\right) = \begin{bmatrix} 0.035 \\ 0.004 \\ 6 \end{bmatrix} \begin{matrix} \text{m} \\ \text{m} \end{matrix} \qquad f\left(d_{b_1}\right) = -0.001$$

False Position (still hidden)

$$d_{fp} := falsepos\left(f, 0.0, 0.5, \varepsilon_{tolerance}\right) = \begin{bmatrix} 0.033 \\ 0.002 \\ 2 \end{bmatrix} \begin{matrix} \text{m} \\ \text{m} \end{matrix} \qquad f\left(d_{fp_1}\right) = 2.579 \cdot 10^{-4}$$

Secant Method

$$d_s := secant\left(f, 0.0, 0.5, \varepsilon_{tolerance}\right) = \begin{bmatrix} 0.034 \\ 1.953 \cdot 10^{-4} \\ 3 \end{bmatrix} \begin{matrix} \text{m} \\ \text{m} \end{matrix} \qquad f\left(d_{s_1}\right) = 2.19 \cdot 10^{-6}$$

$$secant\left(f, 0.0, 0.1, \varepsilon_{tolerance}\right) = \begin{bmatrix} 0.034 \\ 0.003 \\ 2 \end{bmatrix} \begin{matrix} \text{m} \\ \text{m} \end{matrix} \qquad secant\left(f, 0.4, 0.5, \varepsilon_{tolerance}\right) = \begin{bmatrix} 0.034 \\ 0.001 \\ 2 \end{bmatrix} \begin{matrix} \text{m} \\ \text{m} \end{matrix}$$

Newton-Raphson Method

$$d_n := newton\left(f, 0, \varepsilon_{tolerance}\right) = \begin{bmatrix} 0.034 \\ 1.177 \cdot 10^{-4} \\ 4 \end{bmatrix} \begin{matrix} \text{m} \\ \text{m} \end{matrix} \qquad f\left(d_{n_1}\right) = -3.082 \cdot 10^{-8}$$

$$d_n := newton\left(f, 0.5, \varepsilon_{tolerance}\right) = \begin{bmatrix} 0.034 \\ 2.777 \cdot 10^{-4} \\ 2 \end{bmatrix} \begin{matrix} \text{m} \\ \text{m} \end{matrix} \qquad f\left(d_{n_1}\right) = -1.708 \cdot 10^{-7}$$

And for fun, and teachable moments, let's try our original backets to see how far

$$d_{newton1} := 0 - \frac{f(0)}{f'(0)} = 4.032 \cdot 10^{10} \quad \text{m} \qquad d_{newton1} := 0.5 - \frac{f(0.5)}{f'(0.5)} = 0.046 \quad \text{m}$$

EEK! Look at the one with 0! let's look at the denominator!
Since the slope is shallow, it sends the answer WELL away from our desired root.

$$f'(0) = -5.159 \cdot 10^{-13}$$

Indexing Starts at 1
Default Units: SI

Now let's try our solve block approach.
Note the heavy commenting and use of units!

Guess Values

$K_s := 3.467 \cdot 10^{-5}\ m \cdot s^{-1}$ Silt Loam Saturated Hydraulic Conductivity

$\psi_s := 21.8 \cdot cm$ Silt Loam Saturated Matric ("Suction") Potential

$\Theta_s := 0.435$ Silt Loam Saturation Water Content

$\Theta := 0.400$ Ambient Unsaturated Soil Water Content

$\Delta\Theta := \Theta_s - \Theta$ Wetting Front Deficit

$t := 10\ min$ Time into infiltration process

$d := 0.01\ cm$ First-Guess for depth penetration of Wetting Front

Constraints

$K_s \cdot t + \psi_s \cdot \Delta\Theta \cdot ln\left(1 + \frac{d}{\psi_s \cdot \Delta\Theta}\right) - d = 0$ Green-Ampt Equation

$d > 0$ Set all valid depths to be above zero (depth points downward)

Solver

$d_{block} := \mathbf{find}(d) = 0.034\ m$ Solver for depth

Testing the solution - it should be a value of zero...

$$K_s \cdot t + \psi_s \cdot \Delta\Theta \cdot \ln\left(1 + \frac{d_{block}}{\psi_s \cdot \Delta\Theta}\right) - d_{block} = 0\ m$$

And finally get that infiltration rate

$$infiltration\_rate\_at\_time\_t := K_s \cdot \left(\frac{\psi_s \cdot \Delta\Theta}{d_{block}} + 1\right) = 0.255\ \frac{cm}{min}$$